DMTree Artifact

This repository contains the artifact for the paper "**DMTree: Towards Efficient Tree Indexing on Disaggregated Memory via Compute-side Collaborative Design**", accepted at USENIX FAST'26.

Artifact Overview

This artifact allows reviewers to reproduce the main experimental results of the paper, including:

- DMTree outperforms existing state-of-the-art range indexes on DM for both point operations (i.e., searches, inserts, and updates) and range operations (i.e., scans) under various workloads (e.g., Micro-benchmarks and YCSB workloads).

We evaluate and compare **DMTree** with five state-of-the-art DM-optimized range indexes: **Sherman, ROLEX, SMART, dLSM, and CHIME**, all evaluated based on their open-source implementations.

- Sherman *(SIGMOD'22)* is a DM-optimized B+-tree.
- ROLEX *(FAST'23)* is the latest learned index on DM.
- SMART *(OSDI'23)* is the latest radix tree on DM.
- dLSM *(ICDE'23)* is the state-of-the-art DM-optimized LSM-tree.
- CHIME *(SOSP'24)* is a DM-optimized hybrid index that combines the B+ tree with hopscotch hashing.

Cluster setup

- We provide a server cluster consisting of 7 nodes (skv-node1 to skv-node7) for reproducing the experiments in our paper. One node, `skv-node7`, will be made available as the jump host for accessing the cluster and as the main node for running the reproduction scripts.
- **The AEC may submit their SSH public key through a HotCRP comment.** After we have configured the key on our server, node7 will be accessible via the following command:

  ```
  ssh -p 6672 aefast26@222.195.68.87
  ```

- We sincerely apologize that, due to limited cluster resources, we are unable to provide each AEC with an independent execution environment. To prevent potential conflicts caused by concurrent usage, we have adopted an exclusive access notification mechanism in the following steps, ensuring that only one AEC can run experiments at any given time.
- **We would greatly appreciate it if each AEC could notify in the HotCRP comments when starting or finishing using the cluster**, so that other AECs can be promptly informed of the current status of the cluster.

Environment setup (~15 minutes)

We provide scripts to set up the environment for the evaluation, including cloning the code repository and copying and compiling it across multiple cluster nodes. The scripts are tested on Ubuntu 20.04 LTS.

Step 1: Clone the code repository `aefast26`.

- Due to potential network fluctuations, the clone operation might require a few retries to complete successfully.

```
git clone https://github.com/muouim/aefast26.git
```

Step 2: Ensure exclusive access to the cluster by running the locking script `lock_cluster.sh + reviewer ID`. This step prevents concurrent executions by other AECs and ensures consistent resource usage.

```
cd aefast26
bash lock_cluster.sh "reviewer A"
```

Step 3: Run the following script on `skv-node7`, including copying and compiling the code across multiple cluster nodes.

```
bash build_ae.sh
```

In the `build_ae.sh` script, we copy the code to all nodes and perform node-specific configuration and compilation. This includes modifying memory-related parameters for some baselines on memory nodes (which require larger memory sizes) and adjusting compute node-specific settings for certain baselines (e.g., specifying the compute node ID in the test code for dLSM).

To prevent repeated compilation and concurrent execution from disrupting the established environment, we have implemented a tracking and checking mechanism for the environment setup status when running the `build_ae.sh` script:

- If the script has not been executed before and the code has not been copied or compiled, the script will be executed.
- If the script is currently being executed, a message will prompt: "`The script is already running, please wait.`"
- If the script has already been executed, a message will prompt: "`Environment setup is complete, no need to run the script again.`"
- If the AEC wishes to re-run the setup due to unexpected issues or configuration changes, the status can be reset by removing the flag file via:

  ```
  rm /tmp/build_ae.flag
  ```

Evaluations

This section describes how to reproduce the major experiments in our paper. **We suggest running the scripts of 'Micro experiment' first, which can reproduce the main results of our paper while including most of the functionality verification** (i.e., DMTree outperforms existing state-of-the-art range indexes on DM for both point operations (i.e., searches, inserts, and updates) and range operations (i.e., scans)).

Micro experiment

Clean up memory

The `reset_memory.sh` script is used to clear the Linux page cache, ensuring sufficient available memory before running performance experiments. This helps eliminate memory pressure caused by residual cached data from previous runs.

```
bash reset_memory.sh
```

Exp#11-12: Performance with Micro-benchmarks (~11 hours)

We provide this simple experiment to verify our main experimental results quickly: **DMTree outperforms existing state-of-the-art range indexes on DM for both point operations (i.e., searches, inserts, and updates) and range operations (i.e., scans).** Specifically, we preload 1 billion KV pairs and perform 100 million KV operations (including search/insert/update/scan).

You can run this simple experiment via the following command:

```
nohup bash run_simple.sh >run_simple.output 2>&1 &
```

This launches the experiment in the background and redirects the output to a log file, allowing you to safely close the terminal without interrupting the execution.

- You can monitor the execution progress in the `run_simple.output` file, which indicates the current baseline being tested.
- **Once the script has completed all experiments, you will see the message `---------- All phases completed. ----------` at the end of the output file**.

The `run_simple.sh` script executes micro-benchmark experiments across all baseline systems and leverages Python scripts to structure and visualize the results for comparative analysis.

To prevent repeated experiments and concurrent execution from disrupting the running experiments, we have implemented a tracking and checking mechanism for the simple experiment status when running the `run_simple.sh` script:

- If the script has not been executed before and the code has not been copied or compiled, the script will be executed.
- If the script is currently being executed, a message will prompt: "`The script is already running, please wait.`"
- If the script has already been executed, a message will prompt: "`Simple experiment is complete, no need to run the script again.`".
- If the AEC wishes to re-run the simple experiment due to unexpected results or issues, the status can be reset by removing the flag file with:

  ```
  rm /tmp/simple_exp.flag
  ```

Note: If you wish to release the cluster for use by others after completing the current experiment, **please run the command below to update the cluster's release status.**

```
bash unlock_cluster.sh
```

Exp#11-12: Result analysis

The raw experimental results are stored in the `AE/Data` directory. The processed results are organized and written to the files `simple_results_uniform.csv` and `simple_results_zipfian.csv`. For visual comparison, we also generate bar charts saved as `simple_uniform.pdf` and `simple_zipfian.pdf`.

As shown below, we present the processed results and corresponding plots generated by the script.

- The `Index` column represents different baseline systems.
- The `Workload` column indicates the type of workload applied.
- The `Total` column shows the total throughput (in Mops) of each baseline under the specified workload.

simple_results_uniform.csv

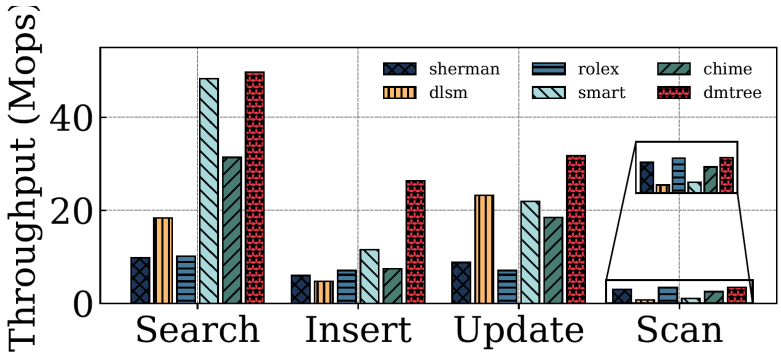
```
Index,Workload,Total,...
dmtree,ycsb-c,49.69586,...
dmtree,insert-only,26.372739999999997,...
dmtree,update-only,31.795920000000002,...
dmtree,scan-only,3.446158,...
fptree,ycsb-c,23.714789999999997,...
fptree,insert-only,13.52334,...
fptree,update-only,13.69716,...
fptree,scan-only,2.654987,...
sherman,ycsb-c,9.81188,...
sherman,insert-only,5.981216,...
sherman,update-only,8.77102,...
sherman,scan-only,2.990589,...
smart,ycsb-c,48.28557,...
smart,insert-only,11.53051,...
smart,update-only,21.919349999999998,...
smart,scan-only,1.041088,...
rolex,ycsb-c,10.17066,...
rolex,insert-only,7.06437,...
rolex,update-only,7.07585,...
rolex,scan-only,3.391118,...
dlsm,ycsb-c,18.36991,...
dlsm,insert-only,4.715047,...
dlsm,update-only,23.23445,...
dlsm,scan-only,0.7661690000000001,...
chime,ycsb-c,31.38408,...
chime,insert-only,7.435649999999999,...
chime,update-only,18.45321,...
chime,scan-only,2.538482,...
```

simple_results_zipfian.csv

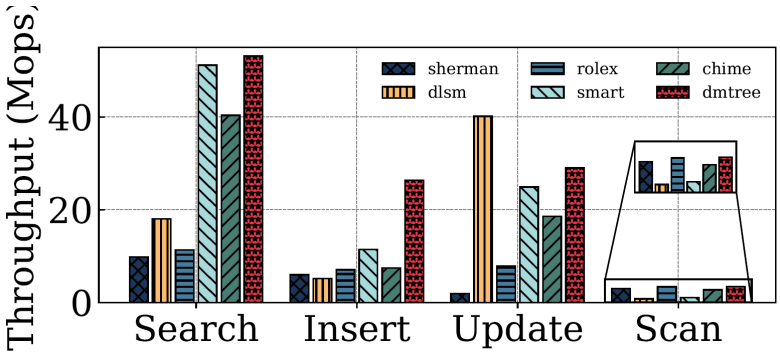
```
Index,Workload,Total,...
dmtree,ycsb-c,53.14693,...
dmtree,insert-only,26.38105,...
dmtree,update-only,29.02227,...
dmtree,scan-only,3.4390020000000003,...
```

```
fptree,ycsb-c,26.38646,...
fptree,insert-only,13.49708,...
fptree,update-only,14.77921,...
fptree,scan-only,2.6599909999999998,...
sherman,ycsb-c,9.80682,...
sherman,insert-only,5.982340999999999,...
sherman,update-only,1.897112,...
sherman,scan-only,3.014283,...
smart,ycsb-c,51.152860000000004,...
smart,insert-only,11.441650000000001,...
smart,update-only,24.898239999999998,...
smart,scan-only,1.049804,...
rolex,ycsb-c,11.33306,...
rolex,insert-only,7.06801,...
rolex,update-only,7.893560000000001,...
rolex,scan-only,3.3781749999999997,...
dlsm,ycsb-c,18.03255,...
dlsm,insert-only,5.184105,...
dlsm,update-only,40.14341,...
dlsm,scan-only,0.77451,...
chime,ycsb-c,40.37225,...
chime,insert-only,7.4409,...
chime,update-only,18.59494,...
chime,scan-only,2.718992,...
```

The experimental results in `simple_results_uniform.csv` are visualized as bar charts, as shown in the figure `simple_uniform.pdf`.



The experimental results in `simple_results_zipfian.csv` are visualized as bar charts, as shown in the figure `simple_zipfian.pdf`.



The output experimental results correspond to those presented in **Figures 11 and 12** of the original paper. To enable quick experimental verification, we only provide the bottleneck performance—**i.e., the performance of each baseline under each workload at the maximum thread count**.

As shown in the original overall experiment figures, the red boxes highlight the data produced by the Micro experiment, representing the performance of each baseline under each workload at the maximum thread count.

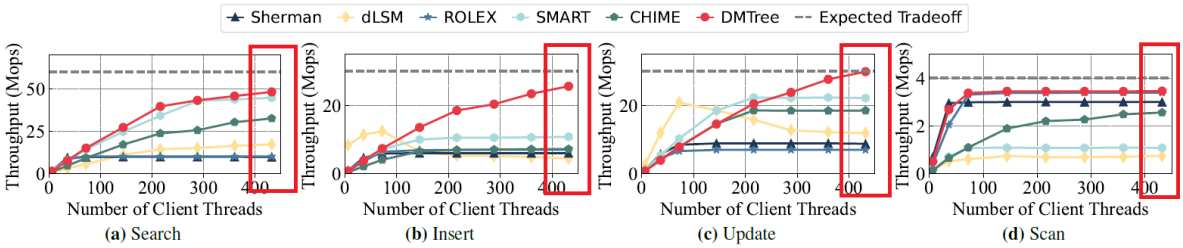


Figure 11: Throughput comparison of range indexes on DM under Micro-benchmarks with Uniform distribution.

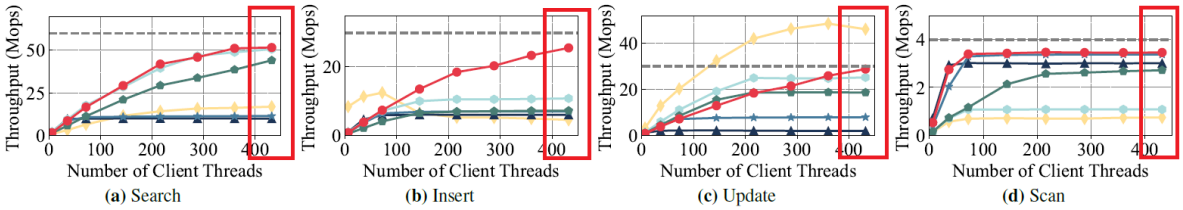


Figure 12: Throughput comparison of range indexes on DM under Micro-benchmarks with Zipfian distribution.

YCSB experiment

We provide this YCSB experiment to verify our overall results: **DMTree outperforms existing state-of-the-art range indexes on DM for both point operations (i.e., searches, inserts, and updates) and range operations (i.e., scans) under various workloads.** Specifically, we preload 1 billion KV pairs and perform 100 million KV operations (including YCSB A/B/C/D/E/F).

Clean up memory

The `reset_memory.sh` script is used to clear the Linux page cache, ensuring sufficient available memory before running performance experiments. This helps eliminate memory pressure caused by residual cached data from previous runs.

```
bash reset_memory.sh
```

Exp#14: Performance with YCSB core workloads (-8 hours)

Running:

You can run this ycsb experiment via the following command:

```
nohup bash run_ycsb.sh >run_ycsb.output 2>&1 &
```

This launches the experiment in the background and redirects the output to a log file, allowing you to safely close the terminal without interrupting the execution.

- You can monitor the execution progress in the `run_ycsb.output` file, which indicates the current baseline being tested.
- Once the script has completed all experiments, you will see the message `---------- All phases completed. ----------` at the end of the output file.

To prevent repeated experiments and concurrent execution from disrupting the running experiments, we have implemented a tracking and checking mechanism for the simple experiment status when running the `run_ycsb.sh` script:

- If the script has not been executed before and the code has not been copied or compiled, the script will be executed.
- If the script is currently being executed, a message will prompt: "`The script is already running, please wait.`"
- If the script has already been executed, a message will prompt: "`YCSB experiment is complete, no need to run the script again.`".
- If the AEC wishes to re-run the YCSB experiment due to unexpected results or other issues, the status can be reset by removing the flag file using:

```
rm /tmp/ycsb_exp.flag
```

Results:

The raw experimental results are stored in the `AE/Data` directory. The processed results are organized and written to the files `ycsb_results_zipfian.csv`. For visual comparison, we also generate bar charts saved as `ycsb_zipfian.pdf`. (To reduce the experiment runtime waiting time for AE, **we have only included Zipfian experiments in the script.** The overall optimization results under the uniform workload are generally similar.)

Note: If you wish to release the cluster for use by others after completing the current experiment, **please run the command below to update the cluster's release status.**

```
bash unlock_cluster.sh
```

Other results:

If AEC wishes to run results with a uniform distribution, please modify the script configurations in each baseline folder. Specifically, update the following in the baseline scripts: `DMTree/script/run_exp14.sh`, `FPTree/script/run_exp14.sh`, `Sherman/script/run_exp14.sh`, `SMART/script/run_exp14.sh`, `ROLEX/script/run_exp14.sh`, `CHIME/script/run_exp14.sh`, `dLSM/script/run_exp14.sh`

Change the `distribution` parameter from "zipfian" to "uniform":

```
workloads="ycsb-a ycsb-b ycsb-c ycsb-d ycsb-e ycsb-f"
threads="72"
distribution="uniform"
```

Reset the YCSB experiment status by removing the flag file using:

```
rm /tmp/ycsb_exp.flag
```

Run the reset memory script:

```
bash reset_memory.sh
```

Start the YCSB experiment script in the background:

```
nohup bash run_ycsb.sh > run_ycsb_2.output 2>&1 &
```